Forward Error Correction Acceleration Proposal for TigerSHARC

The TigerSHARC Digital Signal Processor is being considered as the processor for many 2nd and 3rd Generation Wireless basestations. The TigerSHARC, as it currently exists, makes it an interesting solution. With only a few instruction set changes, however, the TigerSHARC can excel far above the competition for the wireless basestation applications.

The two main sources of competition for the TigerSHARC in the basestation market are competitor DSPs and ASICs. Against competitor DSPs, the TigerSHARC has an architectural advantage but the current clock speed places it about equal to the competitions' overall processing capabilities. This may not be enough to beat the competition. The two main DSP competitors are TI's C6x family and Motorola/Lucents's STARCORE. (STARCORE is at a disadvantage due to its availability date. A big advantage for TI is the existence of the.)

A DSP, rather than an ASIC, is a desirable customer solution because of the software programmability. The downside is no single DSP currently on the market can handle the complete symbol rate processing requirement of the software radio. The processing intensive portion of the software radio is the forward error correction (FEC). Customers are therefore researching a DSP plus ASIC solution or an ASIC only solution to handle the symbol rate processing. The ASIC would execute, at a minimum, the FEC.

The following proposal recommends changes within the instruction set of the current TigerSHARC architecture that will make the TigerSHARC a viable DSP only solution for the software radio. The proposal adds three instructions (TMAX, ACS and PERMUTE) to reduce the cycle requirement for performing the two main FEC algorithms, Viterbi and Turbo decoders.

Summary

Three new TigerSHARC instructions are proposed that will improve the channel decoding performance by a factor of about 4.2x. The first instruction, TMAX, reduces the processing requirement of the Turbo decoder by a factor of three, as shown in Table 1 and Figure 1. It has no effect on the Viterbi decoder. The second instruction, ACS, improves the performance of the Turbo decoder by a factor of 1.3 and the Viterbi decoder by a factor of 1.6. (The turbo decoder improvement is based of the cycle count required if the TMAX instruction is used.)

Algorithm	TS001		TMAX(*)		ACS		PERMUTE	
	Cycles	Efficiency	Cycles	Efficiency	Cycles	Efficiency	Cycles	Efficiency
8-Bit Viterbi	N*91				N*67			
Metric	N*23	6.3%			N*23	6.3%		
Butterfly	N*52	61.5%			N*28	85.7%		
Scale	N*9	N/A			N*9	N/A		
Traceback	N*7	N/A			N*7	N/A		
16-Bit Viterbi	N*128				N*80			
Metric	N*17	17.6%			N*17	17.6%		
Butterfly	N*104	61.5%			N*56	85.7%		
Traceback	N*7	N/A			N*7	N/A		
Turbo Decoder	N*45.50		N*15.50		N*11.50		N*10.75	
Gamma	N*0.50	57.1%	N*0.50	57.1%	N*0.50	57.1%	N*0.50	57.1%
Butterfly	N*24.50	19.2%	N*8.50	55.2%	N*4.50	66.7%	N*4.50	66.7%
Extrinsic	N*20.50	23.0%	N*6.50	72.5%	N*6.50	72.5%	N*5.75	82.2%

Table 1. Iterative TigerSHARC Channel Decoder Performance Resulting from New Instructions

* TMAX improves SNR by 0.5 dB.

** 52.5% efficiency if you include scaling in with Butterfly calculation.

The final instruction, PERMUTE, provides two types of improvement. The first is a cycle reduction. The processing performance is improved by a factor of 1.07. The second benefit is the ability to paramaterize the coding polynomials.

Table 1 summarizes the iterative performance gains from the new instructions on the Turbo and Viterbi decoders. (Efficiency means theoretical cycle count divided by actual cycle count. Theoretical includes adds/subtracts and maximums. Actual includes stalls, memory transfers, data ordering.) Figure 1 shows a performance curve on the Turbo decoder based on the new TigerSHARC instructions.

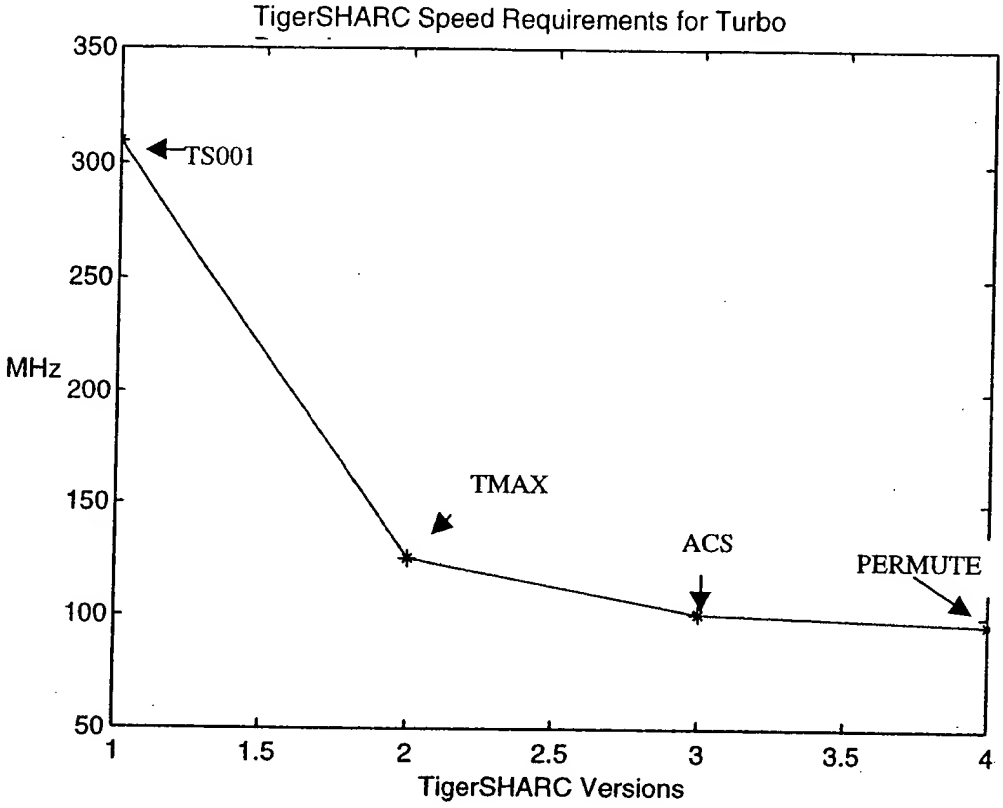


Figure 1. TigerSHARC MIPs requirement to handle 1 384kps data channel for 4 versions.

Software Radio Signal Chain

The following is a high level abstraction of the Software Radio portion of the basestation signal chain. Figure 2 depicts the downlink transmit and the uplink receive. The dotted box labeled FEC represents the forward error correction processing. The dotted box labeled ASIC represents the chip rate processing handled outside of the DSP.

The Chip rate processing is not suitable for today's DSPs, instead an ASIC or FPGA is used. The primary function of the chip rate processor is correlation. Although this can be done in a DSP, it is extremely high rates of simple arithmetic. An ASIC solution makes sense from a cost and performance perspective since there is no need for flexibility.

The Forward error correction block can approach 80% of the symbol rate processing in the software radio. The remaining symbol rate processing functions are implemented in the DSP and most likely customers will want to program them in C. (This requires an efficient compiler.) The goal is to perform the symbol rate processing within the TigerSHARC DSP. Tables 2 and 3 show the MIPs requirement for the TigerSHARC for a complete voice and data channel.

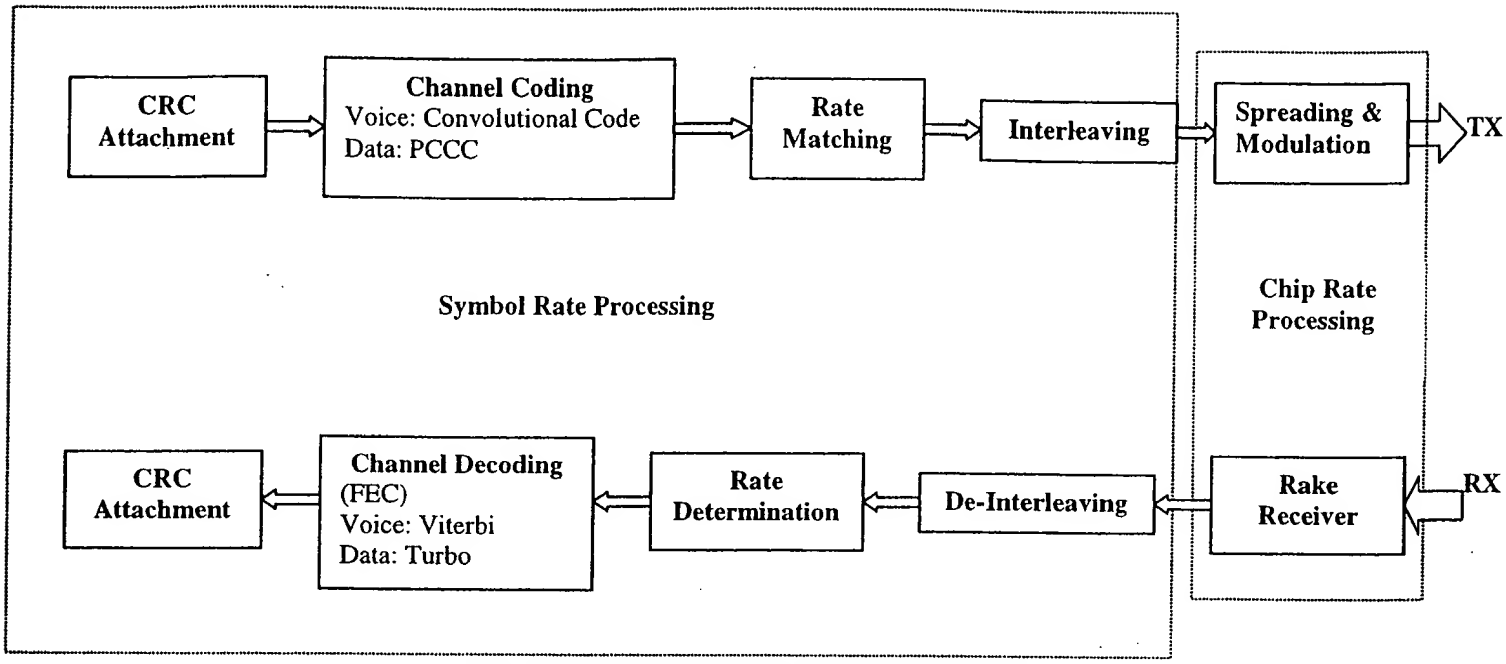


Figure 2. Basestation Software Radio Signal Chain.

User Data Rate	Viterbi(*) Decoder K=9, R=1/3	All Other(*) Functions	Total
12.8 kbps	1.2	1.5	2.7

Table 2. One voice channel MIPs requirement for symbol rate processing.

User Data Rate	Turbo Decoder(*) K=4, R=1/3, 8 iterations	All Other(*) Functions	Total
384 kbps	97	28.6	125.6

Table 3. One data channel MIPs requirement for symbol rate processing.

(*) Benchmark numbers generated from working assembly code
(**) Benchmark numbers from C code estimates

Forward Error Correction Processing

There are two specific techniques, based on the 3GPP specification, to perform FEC in 3G wireless systems. A convolutional coding scheme is specified for voice transmission, and a parallel concatenated convolutional coding scheme (PCCC) is specified for data transmission. The convolutional encoded data is decoded using the Viterbi algorithm and the PCCC encoded data is decoded using a Turbo decoding algorithm. The Turbo and Viterbi decoding schemes are trellis based algorithms.

For the first systems, the customers are requiring a maximum three chip solution for the software radio. The three chips consist of an ASIC for chip rate processing, an ASIC for FEC and a DSP for the remaining processing. Other solutions being entertained are one ASIC and one DSP or just one ASIC to handle the entire signal chain. The processing requirement of one 3 chip solution is 128 12.8Kbps voice channels or 4 384Kbps data channels. Future systems will require handling 1 2Mbps data channel. Based on the data shown in tables 1 and 2 above, it requires one 350 MHz TigerSHARC to handle 128 voice channels (8-bit) and one 505 MHz TigerSHARC to handle 4 data channels with *ZERO* headroom.

Note 1: In order to win, I believe we must supply a "one to two" DSP solution for the entire symbol rate processing signal chain.
Note 2: Our competitive advantage is a software FEC solution.
Note 3: This analysis excludes power and cost parameters. They must also be considered.
Note 4: The rate matching/rate determination, interleaving/ de-interleaving, CRC, and FEC encoding benchmarks in the above tables are based off C code. They may improve, especially if the compiler improves for linear code.

Competitive Analysis

The symbol rate processing, shown in Figure 2, contains three types of algorithms: memory access intensive, quasi memory access intensive and computationally intensive. Memory access intensive means the algorithm does not compute the values of the data, rather it rearranges the data in memory. Quasi memory access intensive algorithms refer to algorithms that compute data values and make high use of memory look up tables to complete the calculations. Computationally intensive algorithms require significant amount of mathematical calculations. Table 4 groups the software radio functions into the three algorithm categories.

Computationally Intensive	Quasi Memory Access	Memory Access
Viterbi Decoder	CRC	Interleaver/ De-Interleaver
Turbo Decoder	Convolutional Encoder	Rate Matching/ Determination
	PCCC	

Table 4. Software Radio Algorithm Processing Categories

The quasi memory and memory access intensive algorithms are mainly sequential algorithms. In only a few cases, such as CRC and Encoders, are the parallel resources of the TigerSHARC utilized. Therefore, from an architectural standpoint, we are only a little better than the C6x and other DSP competition. This, however, is not a concern. The computationally intensive algorithms consume most of the processing time.

The important algorithms to compare competitively are the channel decoders. The Viterbi and Turbo decoder algorithms are extremely computational intensive. There are two approaches to execute these algorithms within the allotted time constraints. One is to create a hardware block with the most basic components and run it as fast as possible. This is the approach used by the TI's C6x DSP. A second approach is to split the processing load across several parallel resources thus reducing clock speed requirements and saving power. The TigerSHARC provides the right amount of parallel resources to efficiently execute the Viterbi and Turbo decoders as specified in the 3GPP standard.

Table 5 shows the channel decoding computation requirements per encoded bit for the TigerSHARC and C6x (Implementations of both Viterbi and Turbo decoders as specified by the 3GPP standard). Appendix A details the calculation of the total MIPs requirement for the symbol rate processing on both processors. The TigerSHARC has a 3.7x architectural advantage over the C6x for Viterbi decoding and a 2x architectural advantage for Turbo Decoding. If we assume all processing except for channel decoding requires the same cycle count for both the C6x and the TigerSHARC, the C6x must run at 978MHz to process 128 voice channels and 937MHz to process 4 data channels as shown in table 4.

Algorithm	TigerSHARC Cycle Count N = # of Bits	TigerSHARC Voice/Data Clock Req[1]	C6x Cycle Count N = # of Bits	C6x Voice/Data Clock Req
8 Bit Viterbi (K=9, R=1/3)	N*91 + 20	345 MHz	N/A	N/A
16 Bit Viterbi (K=9, R=1/3)	N*128 + 21	405 MHz	N*480 [2]	978 MHz
Max Log MAP (K=4, R=1/3)	N*15.5 + 58	505 MHz	N*33.5 [3]	937 MHz

Table 5. Channel Decode benchmark per encoded bit

[1]This column represents the DSP clock speed required to handle the symbol rate processing for 128 voice channels or 4 data channels.
[2]This number was derived from a K=6 Viterbi benchmark on TI's Web Site (66.5*N Cycles).
[3]This number came from a paper presented at DSP World by Jelena Nikolic-Popovic from Texas Instruments

Instruction Proposal

(B/S/L)Rs(d) = TMAX((Rm(d), Rn(d))

Function: TMAX(a,b) = MAX(a,b) + $\ln(1 + e^{-|a-b|})$ where $\ln(1 + e^{-|a-b|})$ is implemented using a 256 element look up table.

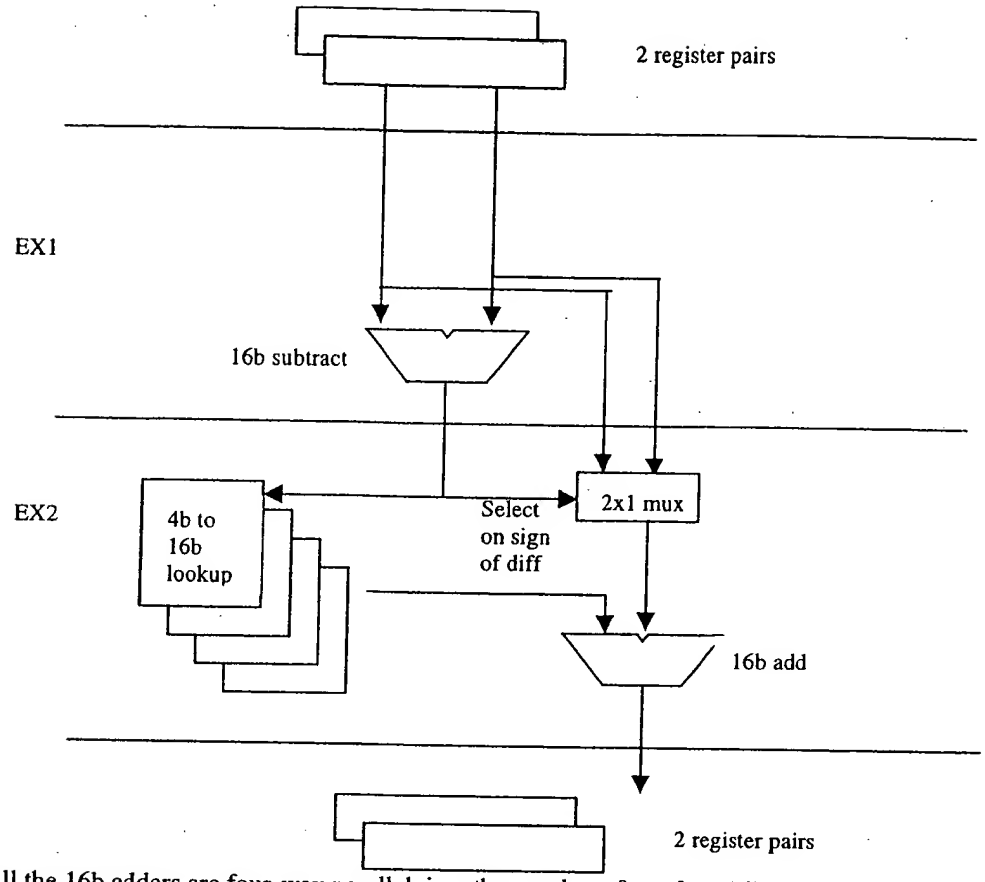
Benefit: Several implementations of Turbo decoders have been published. The two most interesting due to accuracy and processing requirements are the Log MAP and the Max Log MAP. The most accurate and most desired by customers is the Log MAP implementation. The easiest to compute is the Max Log MAP implementation. There is a 0.5 dB performance difference between the two implementations is approximately a 3x computation difference.

The Turbo decode algorithm implemented to achieve the benchmark shown in Table 4 is the Max Log MAP algorithm. The Max Log MAP uses MAX(a,b) as an estimate of $\ln(e^a + e^b)$. The estimate is derived from the equality $\ln(e^a + e^b) = MAX(a,b) + \ln(1 + e^{-|a-b|})$. This estimate results in a 0.5 dB loss in performance.

The Log MAP Turbo decode implementation, required by the customers, adds the correction factor $\ln(1 + e^{-|a-b|})$ to the MAX(a,b). This produces an exact result. The correction factor $\ln(1 + e^{-|a-b|})$ must be implemented using a look-up table. The look-up table requires approximately 17 additional cycles per bit at a minimum. This would change the n*18.5 cycles for Turbo Decoder to 35.5 cycles per bit.

The TMAX instruction enables a TigerSHARC implementation of the Log MAP Turbo decoder without additional cycles. I.e. a 0.5 dB performance gain without a penalty in cycle count.

Implementation: Provide a 16-bit hardware look-up table within the compute block. The look-up table contains up to 16 16 bit values. (Research is required to find the optimum size and content of the look-up table.



All the 16b adders are four-way parallel, i.e., they each perform four 16b adds.

The table look up is a 4b-in to 16-bit out function, that is also four-way parallel. Although called look up table, it can be a simple 2-level combinatorial of approx 1200 gates total. A simple implementation has fixed values. A better implementation scales the output values according to a memory mapped register, but requires more gates.

The longest path is in EX2, with a 16b adder serial with the table lookup (muxing is in parallel with table lookup). The path through the look up table is no more than 2 gates.

(B/S)Rs(q) = ACS((Rm(d), Rn(d), Rp(d))

Function: The ACS function implements a Trellis butterfly. Rm represents the current metric for low order states, Rn represents the current metric for the high order states, Rp represents the local metric for the butterfly and Rs is the new metric. Low order states are states 0 to N/2-1 where N is the number of states, and the high order states are N/2 to N-1. Figure 3 depicts a Trellis butterfly.

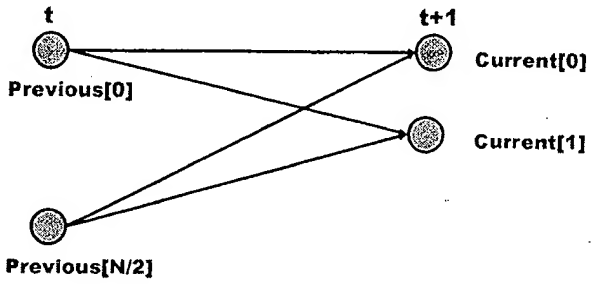


Figure 3. Trellis Butterfly

The following code segment represents the processing involved in a Trellis butterfly with two notable caveats.

sR7:4 = ACS(R5:4, R7:6, R17:16) is:

```
sR9:8    = R5:4 + R17:16, sR1:0 = R5:4 - R17:16;;
sR11:10  = R7:6 + R17:16, sR3:2 = R7:6 - R17:16;;

sR7:4    = MERGE R9:8, R1:0;;
sR3:0    = MERGE R3:2, R11:10;;

sR5:4    = MAX(R1:0, R5:4);;
sR7:6    = MAX(R3:2, R7:6);;
```

Note 1: For Viterbi Trellis, the MAX instruction should be the VMAX instruction. For Turbo Trellis, the MAX instruction should be the TMAX instruction.
Note2: Both Viterbi and Turbo Trellis butterflies can be implemented from the same ACS instruction if the VMAX and TMAX are combined. For Viterbi, the look-up table should have all zero entries and for the Turbo the user will ignore the PR register.

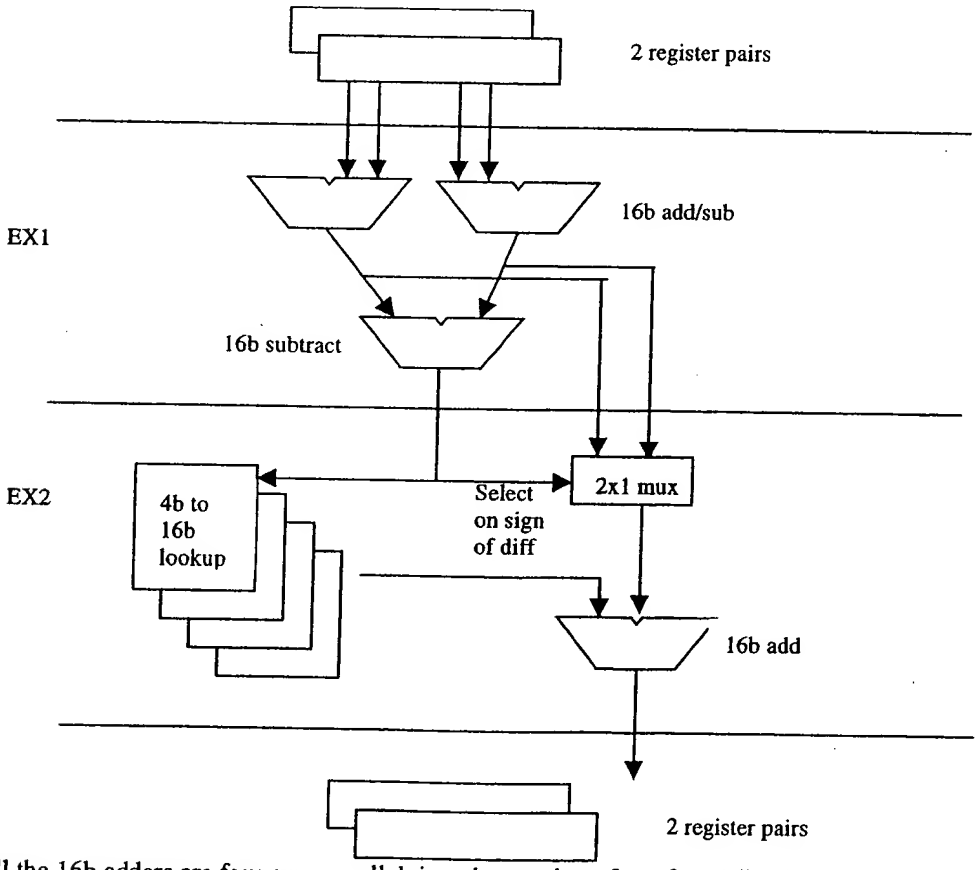
Benefit: Both the Viterbi and Turbo decoders use a Trellis Butterfly implemented by the above code segment. This segment executes 8 16 bit butterflies (4 in X and 4 in Y) and can be easily changed to execute 16 8 bit butterflies. The butterfly executes 6 instructions and 1 stall. The Viterbi can be written to eliminate the stall, however, the turbo decoder does not require enough butterflies to eliminate the stall. Thus, the turbo executes a butterfly in 7 cycles and the Viterbi executes the butterfly in 6 cycles. The 3GPP Viterbi requires the processing of 128 butterflies per bit and the 3GPP turbo decoder requires the processing of 8 butterflies per bit. Thus every cycle below 7 (6) for the new ACS instruction results in a reduction of 1 cycle for the Turbo decoder, 16 cycles for the 16-bit Viterbi decoder and 8 cycles for the 8-bit Viterbi decoder. Table 5 summarizes the performance gain given:

x = # Cycles to perform new ACS instruction.

Algorithm	TigerSHARC	
	Cycle Count N = # of Bits, x = ACS Cycles	Voice/Data Clock Required x = ACS Cycles
8 Bit Viterbi (K=9, R=1/3)	N*(43+8*x)+20	269+131*x MHz
16 Bit Viterbi (K=9, R=1/3)	N*(32+16*x)+21	248+262*x MHz
Max Log MAP (K=4, R=1/3)	N*(8.75+x)+57	333+246*x MHz

Table 5. Channel Decoding benchmarks on TigerSHARC with new ACS instruction

Implementation: The goal is to perform this group of six instructions within 2 to 3 cycles.



All the 16b adders are four-way parallel, i.e., they each perform four 16b adds.

The table look up is a 4b-in to 16-bit out function, that is also four-way parallel. Although called look up table, it can be a simple 2-level combinatorial of approx 1200 gates total. A simple implementation has fixed values. A better implementation scales the output values according to a memory mapped register, but requires more gates.

The longest path is in EX1, with two serial 16b adders, and one level of muxing in between to merge values. (The path through the look up table is no more than 2 gates.)

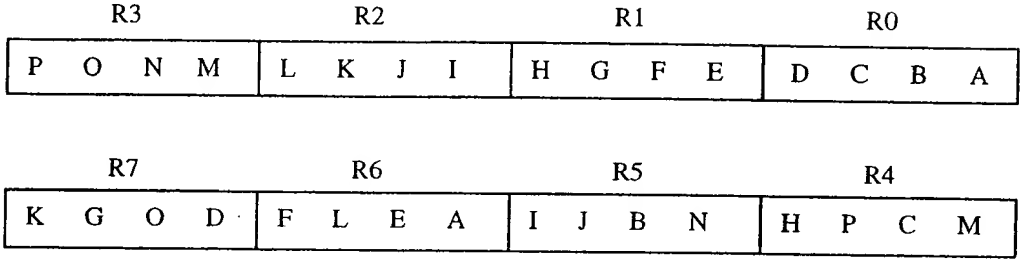
(B/S)Rs(d/q) = PERMUTE Rm(d/q) BY Rn(d)/ immediate

Function: The Permute function implements a word rearrangement within a register group. Rm contains the data in its original order. Rn is an index specifying the new order of the words in Rm.

Example:

bR7:4 = PERMUTE R3:0 BY R9:8;;

R8 = 0x891d7f2c
R9 = 0xa6e35b40



Benefit: The Permute instruction has two very important benefits. The first is a reduction in the cycle count of the current channel decoding implementations on the TigerSHARC. (I do not have an exact count, however, on the Turbo it will be on the order of 1 to 2 n.) The second benefit is it enables the changing of polynomials for the Viterbi and Turbo codes without rewriting the software. The polynomials dictate the order in which many of the Trellis metrics are used. Currently, merges and shifts are to reorder the data. This requires code changes whenever the order must change. If the Permute exists, the order can be parameterized.

Implementation: The permute function may be implemented with a crossbar. The byte version requires a 8x8 crossbar, and the 16-bit version requires a 4x4 crossbar. No unit in the Tigersharc currently has datapath to support this function, so this instruction requires a crossbar to be fully implemented.